CSC263 Assignment #4

You are allowed to work in a group of at most 3 members, and make one submission as a group. You should clearly write names of all group members in the first page of your submission. Everyone in the same group will get the same marks.
Please read and understand the policy on Academic Honesty on the course website. Then, to protect yourself, list on the front of your submission every source of information you used to complete this homework (other than your own lecture and tutorial notes). For example, indicate clearly the name of every student with whom you had discussions, the title and sections of every textbook you consulted (including the course textbook), the source of every web document you used (including documents from the course webpage), etc.
For each question, please write up detailed answers carefully. Make sure that you use notation and terminology correctly, and that you explain and justify what you are doing. Marks will be deducted for incorrect or ambiguous use of notation and terminology, and for making incorrect, unjustified, ambiguous, or vague claims in your solutions.

From this assignment on, most questions on writing algorithms will not explicitly require pseudocode. You may either describe in english, or write pseudocode, or give both (for example, write pseudocode for sub-procedures). When you describe your algorithm, write concisely and precisely in a few lines; you may also itemize the algorithm in several steps.
We will also give some simple exercise questions, which are similar to the ones in the midterm.

1. (*5 points*) Topological sort.

One way to perform topological sorting on a directed acyclic graph (not using DFS) is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph.
(a) (1 point) What happens to this algorithm if the graph has cycles?
(b) (4 points) Explain how to implement this idea so that it runs in time $O(|V| + |E|)$.

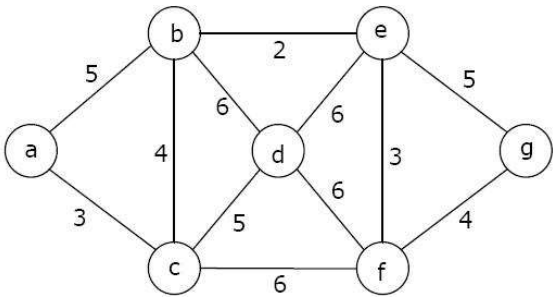
2. (*10 points*) Reachability.

Let $G = (V, E)$ be a directed graph in which each vertex u in V is labeled with a unique integer $L(u)$ from the set $\{1, 2, \ldots, |V|\}$. For each vertex u,

- Let $R(u) = \{v \in V : v \text{ is reachable from } u\}$ be the set of vertices that are reachable from u.
- Define $\min(u)$ to be the vertex in $R(u)$ whose label is minimum, i.e., $\min(u)$ is the vertex v such that $L(v) = \min\{ L(w) : w \in R(u) \}$.

Give an $O(|V| + |E|)$-time algorithm that computes $\min(u)$ for all vertices $u \in V$. Justify the correctness and the running time of your algorithm.

3. (*12 points*) Minimum spanning tree



Find the minimum spanning tree of the graph above using the following (only the final MST is required):

(a) (4 points) Run Prim's algorithm, starting from "a". Whenever there is a choice of vertices, always use alphabetic ordering.
(b) (4 points) Run Kruskal's algorithm. Whenever there is a choice of edges, always use alphabetic ordering of the endpoints. (Also use alphabetic ordering for the two endpoints of each edge.)
(c) (4 points) Give all other MSTs not in (a) and (b). How many MSTs does it have?

4. (28 points) MST properties

The following statements may or may not be correct. In each case, either prove it if it is correct, or give a counterexample if it is wrong. Always assume that the graph $G = (V, E, w)$ is undirected, connected, and weighted.
(a) If all its edge weights are distinct, then the graph has unique MST.
(b) If G has more than $|V| - 1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a MST.
(c) If G has a cycle with a unique heaviest edge, then this edge cannot be part of a MST.
(d) For any edge with minimum weight in G, it must be part of some MST.
(e) If the lightest edge is unique in G, then it must be part of every MST.
(f) If an edge is part of some MST, then it must be a lightest edge crossing some cut of G.
(g) If G has a cycle with a unique lightest edge, the this edge must be part of every MST.

5. (15 points) An algorithm for MST

We will develop a new algorithm for finding minimum spanning trees in undirected, connected, weighted graphs. It is based on the following property:

> Pick any cycle in the graph, and let e be the heaviest edge in that cycle. Then there is a MST that does not contain e.

(a) (4 points) Prove this property.
(b) (4 points) The new MST algorithm on $G = (V, E, w)$ is as follows:

- Sort the edges by their weights in decreasing order
- For each edge e (picking in decreasing order), if it is part of a cycle, remove it from G
- Return the resulting G

Prove this algorithm is correct.
(c) (4 points) On each iteration, the algorithm must check whether there is a cycle containing a specific edge. Give a linear-time algorithm for this task, and justify its correctness.
(d) (3 points) What is the overall running time of this algorithm, in terms of $|E|$? Justify your answer.

6. (*5 points*) Print set in a disjoint-set forest

Suppose that we wish to add the operation PrintSet(x) , which is given a node x and prints all the members of x's set, in any order. Show how we can add just a single attribute to each node in a disjoint-set forest so that PrintSet(x) takes time linear in the number of members of x's set and the asymptotic running times of the other operations are unchanged. Assume that we can print each member of the set in $O(1)$ time.

7. **(*10 points*) Off-line minimum problem**

The off-line minimum problem asks us to maintain a dynamic set T of elements from the domain $\{1, 2, \ldots, n\}$ under the operations Insert and ExtractMin. We are given a sequence S of n Insert and m ExtractMin

calls, where each key in { 1, 2, … , n } is inserted exactly once. We wish to determine which key is returned by each ExtractMin call. Specifically, we wish to fill in an array *extracted*[1.. m] , where for i = 1 , 2 , … , m , *extracted*[i] is the key returned by the ith ExtractMin call.

The problem is "off-line" in the sense that we are allowed to process the entire sequence S before determining any of the returned keys.

In the following instance of the off-line minimum problem, each operation Insert(i) is represented by the value of i and each ExtractMin is represented by the letter E :

4 , 8 , E , 3 , E , 9 , 2 , 6 , E , E , E , 1 , 7 , E , 5.

The correct values in the *extracted* array is [4 , 3 , 2 , 6 , 8 , 1], corresponding to the E above:

4 , 3 , 2 , 6 , 8 , 1

To develop an algorithm for this problem, we break the sequence S into homogeneous subsequences. That is, we represent S by I_1 , E , I_2 , E , I_3 , … , I_m , E , I_{m+1}, where each E represents a single ExtractMin call and each I_j represents a (possibly empty) sequence of Insert calls. For each subsequence I_j, we initially place the keys inserted by these operations into a set K_j, which is empty if I_j is empty. We then do the following:

```
OffLineMinimum(m, n)
    for i = 1 to n
        determine j such that i in K[j]
        if j != m + 1
            extracted[j] = i
            let l be the smallest value greater than j for which set K[l] exists
            K[l] = K[j] ∪ K[l], destroying K[j]
    return extracted
```

(a) (4 points) Argue that the array extracted returned by OffLineMinimum is correct.

(b) (6 points) Describe how to implement OffLineMinimum efficiently with a disjoint-set data structure. Give a tight bound on the worst-case running time of your implementation.